

# Roles in NIEM 0.2

Georgia Tech Research Institute

February 17, 2006

## Contents

<b>1</b>	<b>Requirements</b>	<b>1</b>
<b>2</b>	<b>Description of Technique</b>	<b>1</b>
<b>3</b>	<b>Options</b>	<b>2</b>
<b>4</b>	<b>Syntax Examples</b>	<b>3</b>
4.1	Instance . . . . .	3
4.2	Type Definition . . . . .	3
<b>5</b>	<b>Advantages</b>	<b>4</b>
<b>6</b>	<b>Disadvantages</b>	<b>4</b>

## 1 Requirements

**Requirement 1** Represent roles, either with an activity, or without an activity

**Requirement 2** Represent multiple roles for the same person (or other base object) at the same time

**Requirement 3** Identify the type of the base object of the role (e.g. this is a role of a person)

## 2 Description of Technique

Make the distinction between something that is a specialization of an object, and something that is a role of an object. A role is an independently valid function of an object. A role may have a life cycle independent of any specific activity. Continue to use type inheritance for specialized objects. Adopt the concept of a role as an object that represents a specific function of another object.

Define the following terms:

**Base object:** Some object defined in the data model

**Role object:** An object that represents a specific function of the base object

**Base object type:** The XML Schema type of the base object

**Role object type:** The XML Schema type of the role object

**RoleOf:** A property of a role object. The RoleOf property specifies the base object, of which the role object is a function.

Refactor the data model to account for roles. This refactoring will consist of the following steps:

- For each type in the data model, determine if it defines a role object.
- If it defines a role object, then:
  - If it is derived (i.e. using `xsd:extension`) from its base object type, then:
    - \* Remove the derivation.
    - \* Replace it with a derivation from the super type.
    - \* Add to it an element `RoleOfReference`, referring to its base object type.
  - If it is derived from an ancestor of its base object type, then:
    - \* Refactor its parent type.

This refactoring will occur along with other proposed refactoring.

### 3 Options

1. The cardinality (`minOccurs`, `maxOccurs`) of the `RoleOf` element in a role object could be defined several ways. Cardinality of (0, `unbounded`) maintains consistency with the GJXDM 3.0, and is the recommended value.

**`minOccurs="0"`:** Allows, for example, an enforcement official to be described without referencing the person of which it is a role.

**`minOccurs="1"`:** Requires the base object to appear. For example, an enforcement official must specify a person of which it is a role.

**`maxOccurs="unbounded"`:** Allows multiple base objects to appear. For example, you could specify that an officer may have been person A or person B, with varying degrees of certainty.

`maxOccurs="1"`: Requires that, at most, one base object may appear for a role object. This would require that an enforcement official may be a role of no more than one person, but does not help accommodate uncertainty.

Use of (`minOccurs="0"`, `maxOccurs="unbounded"`) would keep the role property in line with GJXDM 3.0 definitions: optional and over-inclusive. It would allow special cases (e.g. the base object is unknown, or there are several possible base objects). Such values may be subset to (`minOccurs="1"`, `maxOccurs="1"`), or any other desired cardinality.

2. Roles may be built from common parent types.

For example, there are numerous objects that are roles of a person. We may define a type that is a generic role of a person, and derive specific roles from that type. This may help how some users think about roles. We may decide that roles of a person are a general kind of thing, and that all roles of a person may have common characteristics.

## 4 Syntax Examples

### 4.1 Instance

```
<Person s:id="P1">
  <PersonName>
    <PersonFullName>Fred Smith</PersonFullName>
  </PersonName>
</Person>

<EnforcementOfficial>
  <RoleOfPersonReference s:ref="P1"/>
  <EnforcementOfficialBadgeID>
    <ID>101101</ID>
  </EnforcementOfficialBadgeID>
</EnforcementOfficial>
```

Use of the element `RoleOfPeronsReference` satisfies Requirement 3 , in that it indicates the type of the base object (in this case, a person of type `PersonType`). The type is not enforced by XML Schema validation. It is indicated, and could be enforced by XSLT scripts, but is not enforced by XML Schema validation.

### 4.2 Type Definition

```
<complexType name="EnforcementOfficialType">
  <complexContent>
    <extension base="this:SuperType">
```

```

<sequence>
  <element ref="this:RoleOfPersonReference" minOccurs="0"
    maxOccurs="unbounded"/>
  ...
  ... Additional elements defined for enforcement officials ...
  ...
</sequence>
</extension>
</complexContent>
</complexType>

```

## 5 Advantages

1. Enables accurate modeling of *roles* of a thing versus specialized versions of that thing
2. Enables a thing with multiple roles to be well represented.

Previously, such a case would have required one of two methods:

- (a) Duplication of data and value-based resolution of duplicate objects (i.e. the enforcement official and the victim have the same name, and so must be the same person), or
- (b) Use of the `Relationship` object named `SameAs`, with subject and object pointing to the replicated objects.

## 6 Disadvantages

1. References can't be validated via XML Schema validation. Validation must be conducted via other mechanisms (e.g. in code, or with XSLT).
2. Increases the number of objects: There is a separate object for a person and each role of that person.

Given  $n$  roles for a person, there would be  $n + 1$  objects, while the previous method would only require  $n$  objects.

3. Requires an ID-dereferencing step to determine properties of a person.

For example, trying to determine the name of an enforcement official requires first finding the person object, then using the name of that person.